REAL-TIME CHAT & COLLABORATION TOOL

Internship Project Report
Submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

in the subject of
Full Stack Web Development

Submitted By

S No.	Name	Registration No.
1.	Kailash Chandra	12211816
2.	Prashant Rana	12215345
3.	Vinit Kumar Singh	12218172

Under the Guidance of

Project Guide:
Designation:

Department of Computer Science & Engineering

Lovely Professional University

Academic Year: 2025 – 2026

1. Problem Statement

1.1 Real-World Problem Definition

Today, instant communication isn't just a convenience—it's an expectation, especially for students, teams, and organizations working to accomplish real tasks. Traditional methods like email or any messaging system that isn't immediate only serve to slow things down. When people can't exchange ideas or make decisions quickly, projects get delayed and momentum is lost. On top of that, communication isn't limited to text anymore. Now, everyone wants to share images, audio, and videos, all within the same conversation.

That's where the Real-Time Chat & Collaboration Tool comes in. The objective: create a web-based application that enables users to send text and media instantly, right from their browsers, using a modern full-stack framework. The entire application runs locally but follows industry-standard architecture closely. In essence, it strives to provide a professional, seamless user experience—even within a smaller, contained environment.

1.2 Industry Relevance

Real-time chat and collaboration tools are everywhere today. Software development teams depend on them to resolve issues and deliver features quickly. Educational institutions use them for group projects and rapid feedback. Companies with distributed workforces rely on these tools, and customer support services couldn't function without instant messaging.

Platforms like Slack, Microsoft Teams, and Discord have set the benchmark for what users expect: real-time messaging, media sharing, and a smooth, well-designed interface. This project draws inspiration from those platforms and distills their essential features into a focused, local solution that maintains high standards of quality and usability.

1.3 Target Users

This tool is designed for college students collaborating on assignments, small project teams seeking a quick communication hub, online communities in need of an easy chat platform, and educational institutions aiming to keep conversations fast and adaptable.

2. Requirement Analysis

2.1 Functional Requirement

Each user selects a unique username to enter the chat. Once inside, they can send and receive messages instantly—no lag, just real-time interaction. Media sharing is straightforward; users can upload images, audio, or videos and share them directly in the chat. These files are stored in MinIO's local object storage, and users can access them through object URLs.

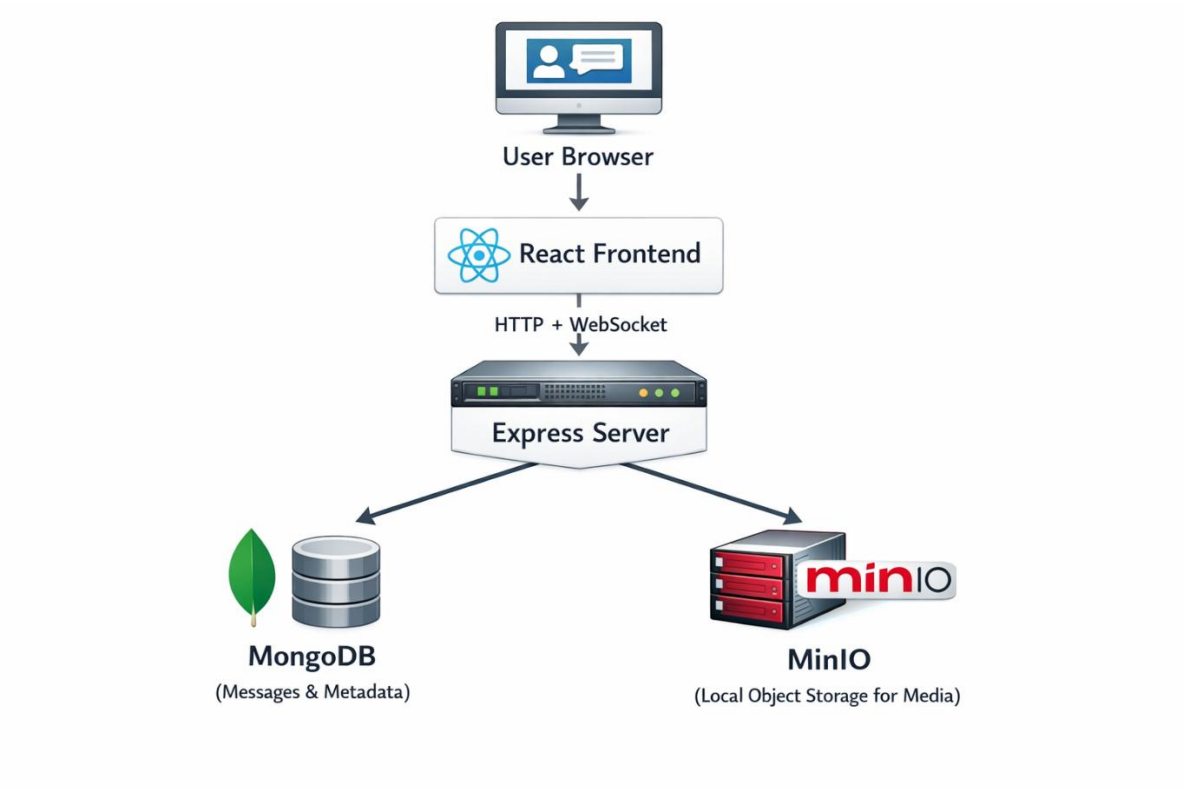
A live list displays which users are online at any given time. Whenever someone joins or leaves, the chat notifies all users immediately. The application saves the entire chat history and records media file information in the database.

2.2 Non-Functional Requirements

Messages must be exchanged with minimal delay—low latency is essential. The system supports many users simultaneously and maintains high performance regardless of load. Data is transferred securely and reliably between users. The architecture is designed to scale as user numbers increase.

The interface is smooth and user-friendly on both desktop and mobile devices. It operates seamlessly across all major browsers. Even if the server is restarted, all data remains intact—storage persists through any interruptions.

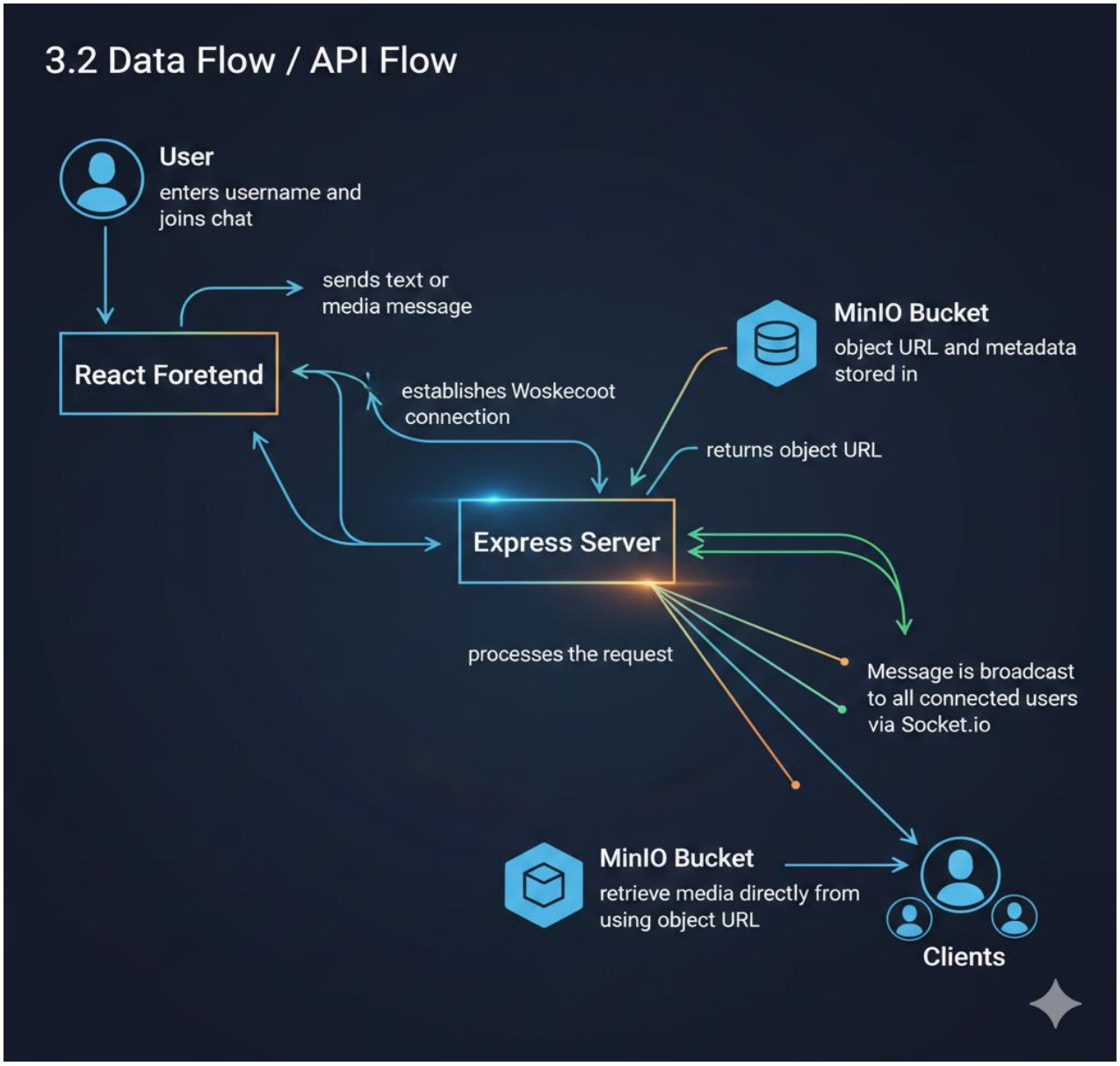
3. System Architecture



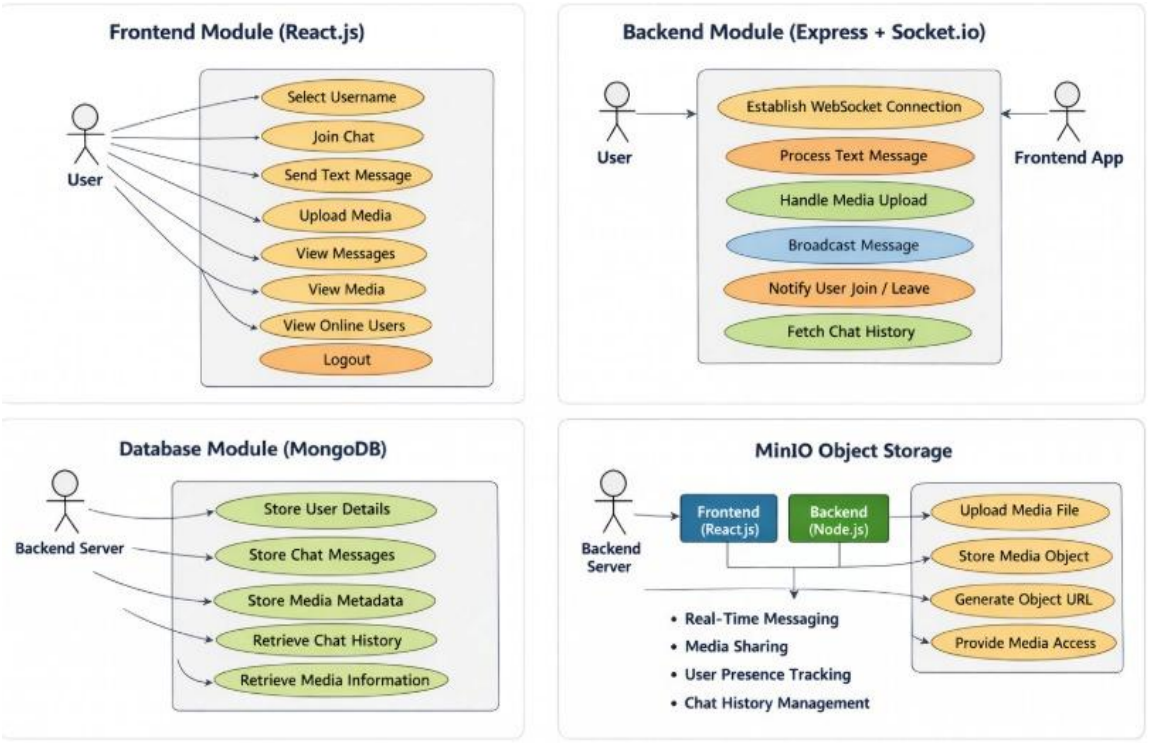
3.1 High-Level Architecture

The system utilizes a simple client-server model, all operating on localhost.

The frontend is built with React.js, providing the user interface. The backend uses Express.js, which handles both APIs and real-time communication. MongoDB stores messages and related metadata, while media files are saved in MinIO object storage. Real-time messaging is facilitated by Socket.io.



3.2 Data Flow / API Flow



i. User enters username and joins chat

ii. React frontend establishes WebSocket connection

iii. User sends text or media message

iv. Express server processes the request

v. Media file is uploaded to MinIO bucket

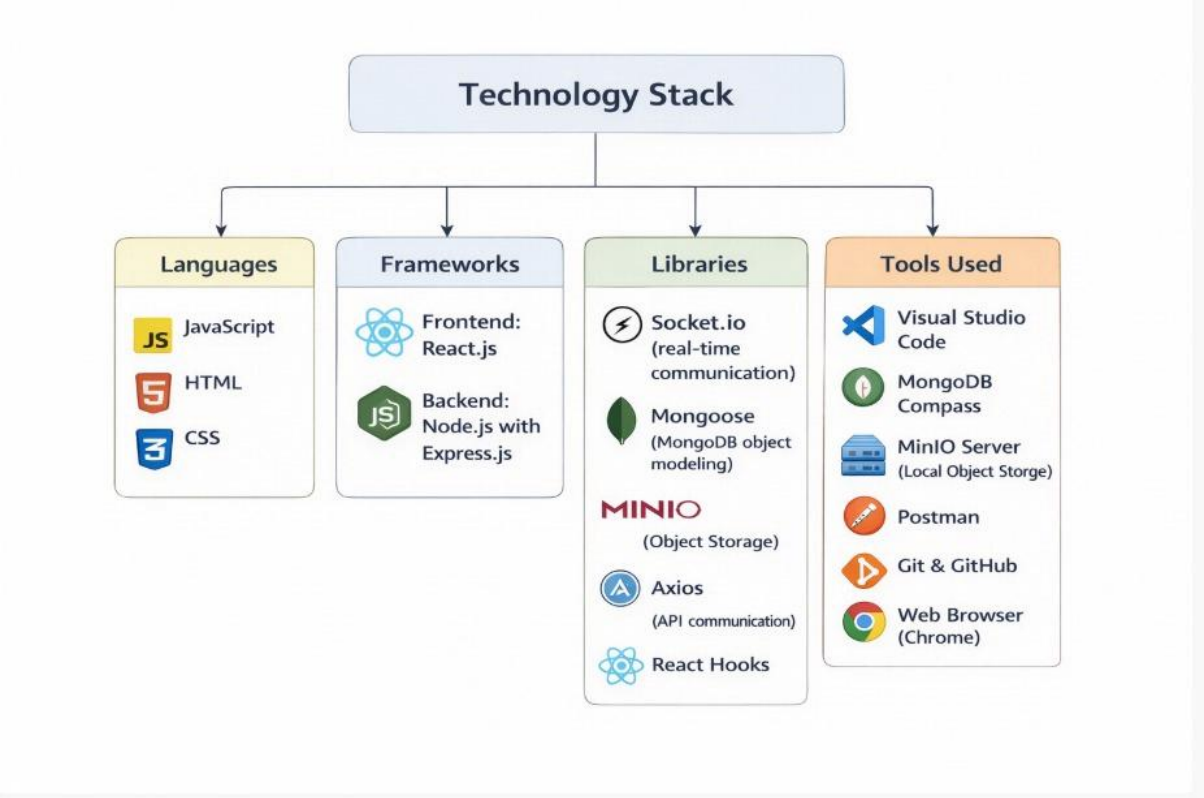
vi. MinIO returns object URL

vii. Object URL and metadata stored in MongoDB

viii. Message is broadcast to all connected users via Socket.io

ix. Clients retrieve media directly from MinIO using object URL

4. Technology Stack



4.1 Languages

- JavaScript
- HTML
- CSS

4.2 Frameworks

- Frontend**:** React.js
- Backend**:** Node.js with Express.js

4.3 Libraries

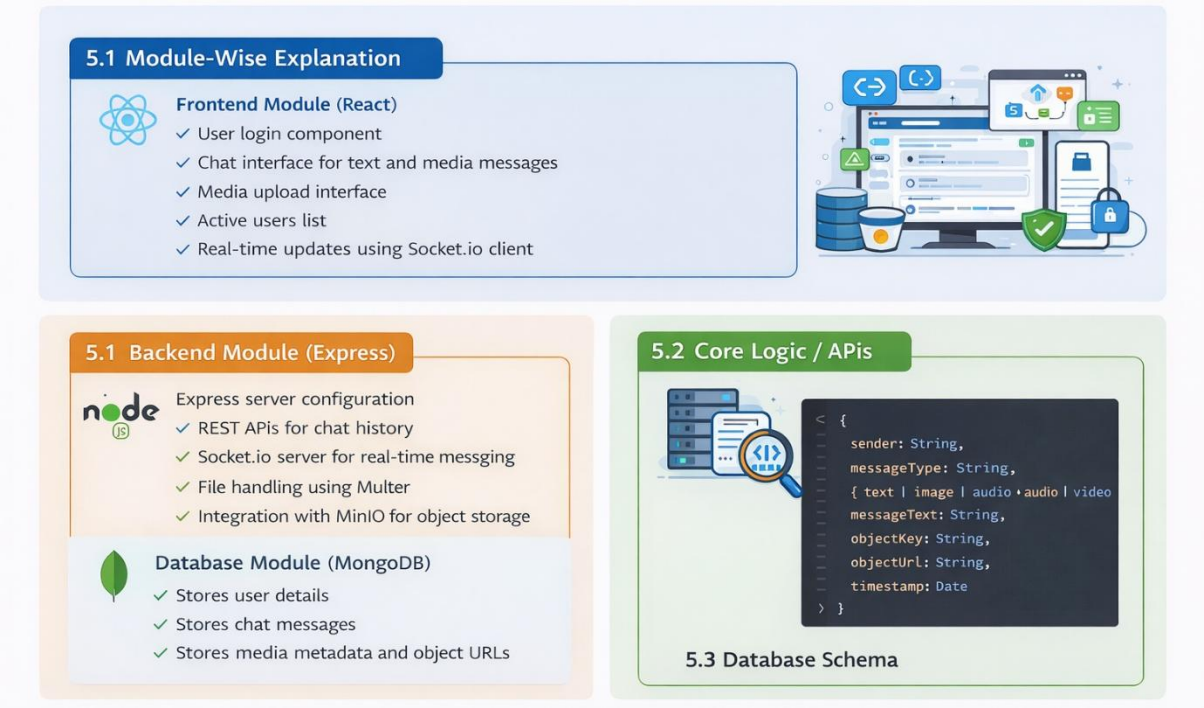
- Socket.io (real-time communication)
- Mongoose (MongoDB object modeling)
- MinIO (Object Storage)
- Axios (API communication)
- React Hooks

4.4 Tools Used

- Visual Studio Code
- MongoDB Compass
- MinIO Server (Local Object Storage)
- Postman

- Git & GitHub
- Web Browser (Chrome)

5. Implementation



5.1 Module-Wise Explanation

Frontend Module (React)

- User login component
- Chat interface for text and media messages
- Media upload interface
- Active users list
- Real-time updates using Socket.io client

Backend Module (Express)

- Express server configuration
- REST APIs for chat history
- Socket.io server for real-time messaging
- File handling using Multer
- Integration with MinIO for object storage

Database Module (MongoDB)

- Stores user details
- Stores chat messages
- Stores media metadata and object URLs

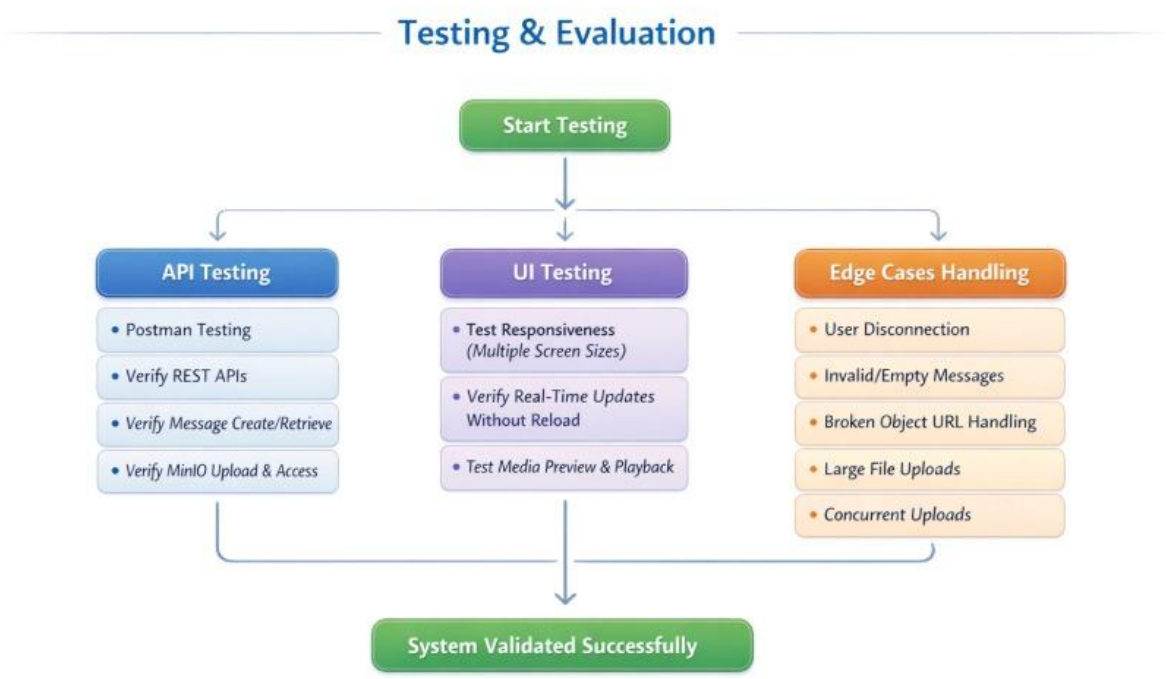
5.2 Core Logic / APIs

- WebSocket-based real-time communication
- Event-driven message broadcasting
- Secure media upload pipeline
- Object storage using MinIO buckets
- Separation of media storage and metadata storage

5.3 Database Schema

```
{
  sender: String,
  messageType: String,   // text | image | audio | video
  messageText: String,
  objectKey: String,
  objectUrl: String,
  timestamp: Date
}
```

6. Testing & Evaluation



6.1 API Testing

- Tested REST APIs using Postman
- Verified message creation and retrieval
- Verified MinIO upload and access APIs

6.2 UI Testing

- Tested responsiveness on different screen sizes
- Verified real-time updates without page reload
- Tested media preview and playback

6.3 Edge Cases Handled

- Sudden user disconnection
- Invalid or empty messages
- Broken object URL handling
- Large file upload handling
- Concurrent uploads by multiple users

7. Deployment

7.1 Deployment Type

- Fully deployment on cloud
- Optional local deployment without changing architecture

7.2 Screenshots / Demo

- Chat interface screenshots
- Real-time text and media sharing demonstration

8. Outcome & Learnings

8.1 What Worked

- Smooth real-time messaging
- Efficient media storage using MinIO
- Clean separation between database and object storage
- Stable full-stack integration

8.2 Challenges Faced

- Managing WebSocket connections
- Handling large media uploads
- Synchronizing media URLs with real-time events

8.3 Future Enhancements

- User authentication and authorization
- Private chat rooms
- Signed URLs for secure media access
- Media compression
- Distributed MinIO setup
- Voice and video calling
- Adding AI features

Conclusion

The Real-Time Chat & Collaboration Tool successfully demonstrates a modern full-stack application using React, Express, MongoDB, and MinIO. The project follows industry-grade architecture while operating completely on localhost, making it ideal for academic evaluation as well as practical learning.